

AD-A222 909

DTIC FILE COPY

2

Symbolic architectures for cognition

Allen Newell, Paul S. Rosenbloom, and John E. Laird

Technical Report AIP - 62

1989

Allen Newell

Departments of Computer Science and Psychology
Carnegie Mellon University

Paul S. Rosenbloom

Information Sciences Institute
University of Southern California

John E. Laird

Department of Electrical Engineering and Computer Science
University of Michigan

DISTRIBUTION STATEMENT A

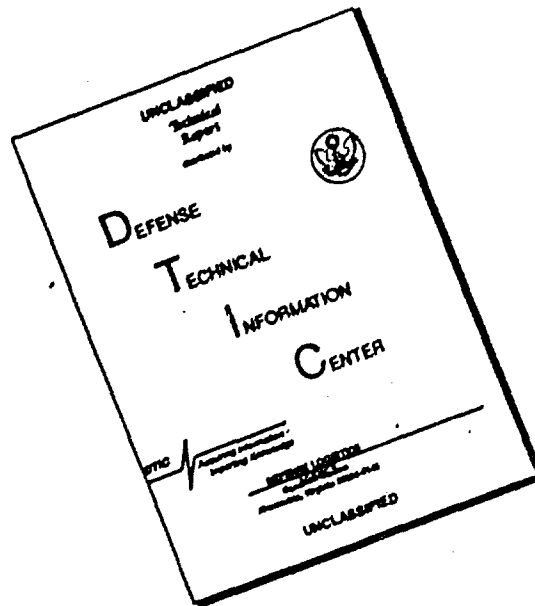
Approved for public release;
Distribution Unlimited

To appear in M. Posner (Ed.), *Foundations of Cognitive Science*, MIT Bradford Books, Cambridge, MA, 1989 (in press).

This research was supported by the Computer Sciences Division, Office of Naval Research, under contract number N00014-86-K-0678. Reproduction in whole or in part is permitted for any purpose of the United States Government. Approved for public release; distribution unlimited.

90 06 19 003

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; Distribution unlimited		
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AIP - 62			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION Carnegie-Mellon University		6b. OFFICE SYMBOL (if applicable)	7a. NAME OF MONITORING ORGANIZATION Computer Sciences Division Office of Naval Research		
6c. ADDRESS (City, State, and ZIP Code) Department of Psychology Pittsburgh, Pennsylvania 15213			7b. ADDRESS (City, State, and ZIP Code) 800 N. Quincy Street Arlington, Virginia 22217-5000		
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Same as Monitoring Organization		8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-86-K-0678		
8c. ADDRESS (City, State, and ZIP Code)			10. SOURCE OF FUNDING NUMBERS p4000ub201/7-4-86		
PROGRAM ELEMENT NO. N/A		PROJECT NO. N/A	TASK NO. N/A	WORK UNIT ACCESSION NO. N/A	
11. TITLE (Include Security Classification) Symbolic architectures for cognition					
12. PERSONAL AUTHOR(S) Allen Newell, Paul S. Rosenbloom, and John E. Laird					
13a. TYPE OF REPORT Technical		13b. TIME COVERED FROM 86Sept15 TO 91Sept14		14. DATE OF REPORT (Year, Month, Day) 1989	
15. PAGE COUNT 39					
16. SUPPLEMENTARY NOTATION To appear in M. Posner (Ed.), <u>Foundations of Cognitive Science</u> , MIT Bradford Books, Cambridge, MA, 1989 (in press).					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Cognitive science, artificial intelligence, symbolic architectures, Act*, Soar, 16		
19. ABSTRACT (Continue on reverse if necessary and identify by block number) SEE REVERSE SIDE					
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT <input type="checkbox"/> UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION		
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Alan L. Mayrowitz			22b. TELEPHONE (Include Area Code) (202) 696-4302		22c. OFFICE SYMBOL N00014

Abstract

This chapter treats the architecture, which is the fixed structure that provides the frame within which cognitive processing in the mind takes place. It describes what an architecture is and how it enters into cognitive theories of the mind. It concentrates on symbolic architectures, the family that includes the architectures central to computer science. It does not treat foundational matters or connectionist architectures. After treating in detail the general requirements of a cognitive architecture, it uses Act* and Soar, two architectures relevant to the study of human cognition, to illustrate matters in detail. *Keywords:*

40, 48



Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	23

3

Symbolic Architectures for Cognition

Allen Newell, Paul S. Rosenbloom, and
John E. Laird

In this chapter we treat the fixed structure that provides the frame within which cognitive processing in the mind takes place. This structure is called the *architecture*. It will be our task to indicate what an architecture is and how it enters into cognitive theories of the mind. Some boundaries for this chapter are set by the existence of other chapters in this book. We will not address the basic foundations of the computational view of mind that is central to cognitive science, but assume the view presented by Pylyshyn in chapter 2. In addition to the groundwork Pylyshyn also deals with several aspects of the architecture. Chapter 1, the overview by Simon and Kaplan, also touches on the architecture at several points and also in the service of the larger picture. Both treatments are consistent with ours and provide useful redundancy.

This chapter considers only *symbolic architectures* and, more particularly, architectures whose structure is reasonably close to that analyzed in computer science. The space of all architectures is not well understood, and the extent and sense to which all architectures must be symbolic architectures is a matter of contention. Chapter 4 by Rumelhart covers nonsymbolic architectures, or more precisely the particular species under investigation by the connectionists.

First, we sketch the role the architecture plays in cognitive science. Second, we describe the requirements the cognitive architecture must meet. Third, we treat in detail the nature of the cognitive architecture. Fourth, we illustrate the concepts with two cognitive architectures, Act* and Soar. Fifth, we indicate briefly how theories of the architecture enter into other studies in cognitive science. We close with some open questions.

3.1 The Role of the Architecture in Cognitive Science

Viewing the world as constituted of *systems of mechanisms* whose behavior we observe is part of the common conceptual apparatus of science. When the systems are engineered or teleological, we talk about *structure* and *function*—a system of a given structure producing behavior that

MASTER SET
F 10 10 10

performs a given function in the encompassing system. The term *architecture* is used to indicate that the structure in question has some sort of primary, permanent, or originating character. As such one can talk about the architecture of the mind or a part of the mind in a general and descriptive way—the architecture of the visual system, an architecture for the conceptual system, and so on.

In cognitive science the notion of architecture has come to take on a quite specific and technical meaning, deriving from computer science. There the term stands for the hardware structure that produces a system that can be programmed. It is the design of a machine that admits the distinction between hardware and software.¹ The concept of an architecture for cognitive science then is the appropriate generalization and abstraction of the concept of computer architecture applied to human cognition: the fixed system of mechanisms that underlies and produces cognitive behavior. As such, an appropriate starting place is a description of an ordinary computer architecture.

The Architecture of Computers

Consider a simple (uniprocessor) digital computer. The top of figure 3.1 shows the gross physical configuration of the system. There is a set of components—a processor, a primary memory, and so on—joined by communication links (the link connecting almost everything together is called the bus). Over the links flow streams of bits. A look inside the processor (the lower-left portion of the figure) reveals more anatomical detail of components connected by links—a number of register memories; a data unit for carrying out various operations such as addition, intersection, shifting, and the like; and an interpreter unit for carrying out the instructions of the program. The primary memory contains a few instructions from a program. The address in the program address register points to one of them, which is brought into the program register and decoded. The left part of the instruction is used to select one of the basic operations of the computer, and the right part is used to address a cell of the primary memory, whose content is retrieved to become an argument for the operation. The repeated acts of obtaining the next instruction, interpreting it, and performing the operation on the argument are called the *fetch-execute cycle*.

Figure 3.1 specifies the architecture of a digital computer (given literary license). It describes a mechanistic system that behaves in a definite way. The language used to describe it takes much for granted, referring to registers, links, decoders, and so on. These components and how they operate require further specification—to say how they are to be realized in circuit technology and ultimately in electron physics—all of which can be taken for granted here.

The behavior of this machine depends on the program and data stored in the memory. Indeed the machine can exhibit essentially any behavior whatsoever depending on the content. By now we are all well ac-

1 LINE SH
REGU
1 LINE L

MASTER SET
Figure 3.1

Size: 27p0×31p0

Figure 3.1 Structure of a simple digital computer

1 LINE SHORT
REGULAR
1 LINE LONG

JS18 17265SSSS6 U3 04-11-89 09-58-39 Page 95

MASTER SET
Pls. & Return

quainted with the amazing variety of such programs—computing statistics, keeping inventories, playing games, editing manuscripts, running machine tools, and so on—but also converting the machine's interaction with its environment to occur by a wide variety of languages, graphic displays, and the like. All this happens because of three things taken jointly: the computer architecture, which enables the interpretation of programs; the flexibility of programs to specify behavior, both for external consumption and for creating additional programs to be used in the future; and lots of memory to hold lots of programs with their required data so that a wide variety of behavior can occur.

Figure 3.1 is only the tip of the iceberg of computer architectures. It contains the essential ideas, however, and serves to introduce them in concrete form.

The Architecture of Cognition

Figure 3.1 epitomizes the invention of the computer, a mechanism that can exhibit indefinitely flexible, complex, responsive, and task-oriented behavior. We observe in humans flexible and adaptive behavior in seemingly limitless abundance and variety. A natural hypothesis is that systems such as that of figure 3.1 reveal the essential mechanisms of how humans attain their own flexibility and therefore of how the mind works.

A chief burden of chapter 2 (and a theme in chapter 1) is to show how this observation has been transformed into a major foundation of cognitive science. The empirical basis for this transformation has come from the immense diversity of tasks that have been accomplished by computers, including, but not limited to, the stream of artificial intelligence (AI) systems. Compelling force has been added to this transformation from the theory of computational mechanisms (Hopcroft and Ullman 1979), which abstracts from much that seems special in figure 3.1 and shows both the sufficiency and the necessity of computational mechanisms and how such mechanisms relate to systems having representations of their external world (that is, having semantics). The architectures that arise from this theory are called symbolic architectures.

As laid out in chapter 2, the human can be described at different system levels. At the top is the *knowledge level*, which describes the person as having goals and knowing things about the world, in which knowledge is used in the service of its goals (by the principle of rationality). The person can operate at the knowledge level only because it is also a *symbol level* system, which is a system that operates in terms of representations and information-processing operations on these representations. The symbol level must also be realized in terms of some substrate, and the architecture is that substrate defined in an appropriate descriptive language. For computers this turns out to be the *register*

1 LINE
R
1 LINE

transfer level, in which bit-vectors are transported from one functional unit (such as an adder) to another, subject to gating by control bits. For humans it is the neural-circuit level, which currently seems well described as highly parallel interconnected networks of inhibitory and excitatory connections that process a medium of continuous signals. Below that of course are other levels of description—neurons, organelles, macromolecules, and on down.

This arrangement of system levels seems very special—it is after all the eye of the needle through which systems have to pass to be able to be intelligent. Nevertheless there is an immense variety of architectures and an immense variety of physical substrates in which they can be implemented. No real appreciation exists yet for this full double variety or its consequences, except that they are exceedingly large and diverse. It is relatively easy to understand a given architecture when presented, though there may be a fair amount of detail to wade through. However, it is difficult to see the behavioral consequences of an architecture, because it is so overlaid by the programs it executes. And it is extremely difficult to compare different architectures, for each presents its own total framework that can carve up the world in radically different ways. Despite these difficulties cognitive science needs to determine the architecture that underlies and supports human cognition.

The architecture does not by itself determine behavior. The other main contributors are the goal the person is attempting to attain, the task environment within which the person is performing, and the knowledge the person has. The first is not only the knowledge of the conditions or situation desired, but also the commitment to govern behavior to obtain such conditions. The second is the objective situation, along with the objective constraints about how the person can interact with the situation. The third is the subjective situation of the person in relation to the task. The knowledge involved in accomplishing any task is diverse and extensive and derives from multiple sources. These sources include the statement or presenting indications of the task, the immediately prior interaction with the task situation, the long-term experience with analogous or similar situations, prior education including the acquisition of skills, and the socialization and enculturation that provide the background orientation. All these sources of knowledge make their contribution.

The goal, task, and knowledge of course constitute the knowledge-level characterization of a person. The architecture's primary role is to make that possible by supporting the processing of the symbolic representations that hold the knowledge. If it did so perfectly, then the architecture would not appear as an independent factor in the determination of behavior any more than would acetylcholine or sulfur atoms. It would simply be the scaffolding that explains how the actual determinants (task and knowledge) are realized in our physical world.

1 LINE SHORT
REGULAR
1 LINE LONG

But the knowledge-level characterization is far from perfect. As the linguists used to be fond of saying, there can be a large gap between competence and performance. The architecture shows through in many ways, both large and small. Indeed much of cognitive psychology is counting these ways—speed of processing, memory errors, linguistic slips, perceptual illusions, failures of rationality in decision making, interference effects of learned material, and on and on. These factors are grounded in part in the architecture. Aspects of behavior can also have their source in mechanisms and structure defined at lower levels—neural functioning, properties of muscle, imperfections in the corneal lens, macromolecular structure of drugs, the effects of raised temperature, and so on. When the architecture fails to support adequately knowledge-based goal-oriented behavior, however, it gives rise by and large to characteristics we see as psychological. Viewed this way, much of psychology involves the investigation of the architecture.

What the notion of the architecture supplies is the concept of the *total* system of mechanisms that are required to attain flexible intelligent behavior. Normally psychological investigations operate in isolation, though with a justified sense that the mechanisms investigated (memory, learning, memory retrieval, whatever) are necessary and important. The architecture adds the total system context within which such separate mechanisms operate, providing additional constraints that determine behavior. The architecture also brings to the fore additional mechanisms that must be involved and that have received less attention in experimental psychology, for instance, elementary operations and control. This requirement of integration is not just a pleasant condiment. Every complete human performance invokes most of the psychological functions we investigate piecemeal—perception, encoding, retrieval, memory, composition and selection of symbolic responses, decision making, motor commands, and actual motor responses. Substantial risks are incurred by psychological theory and experimentation when they focus on a slice of behavior, leaving all the rest as inarticulated background.

A theory of the architecture is a proposal for the total cognitive mechanism, rather than for a single aspect or mechanism. A proposed embodiment of an architecture, such as a simulation system, purports to be a complete mechanism for human cognition. The form of its memory embodies a hypothesis of the form of human symbolic specifications for action; the way its programs are created or modified embodies a hypothesis of the way human action specifications are created or modified; and so on (Newell 1987).

To summarize, the role of the architecture in cognitive science is to be the central element in a theory of human cognition. It is not the sole or even the predominant determinant of the behavior of the person, but it is the determinant of what makes that behavior psychological rather than a reflection of the person's goals in the light of their knowl-

1 LINE S
REG
1 LINE

edge. To have a theory of cognition is to have a theory of the architecture.

3.2 Requirements on the Cognitive Architecture

Why should the cognitive architecture be one way or the other? All architectures provide programmability, which yields indefinitely flexible behavior. Why wouldn't one architecture do as well as another? We need to address this question as a preliminary to discussing the nature of the cognitive architecture. We need to understand the requirements that shape human cognition, especially beyond the need for universal computation. The cognitive architecture must provide the support necessary for all these requirements. The following is a list of requirements that could shape the architecture (adapted from Newell 1980):

1. Behave flexibly as a function of the environment
2. Exhibit adaptive (rational, goal-oriented) behavior
3. Operate in real time
4. Operate in a rich, complex, detailed environment
 - a. perceive an immense amount of changing detail
 - b. use vast amounts of knowledge
 - c. control a motor system of many degrees of freedom
5. Use symbols and abstractions
6. Use language, both natural and artificial
7. Learn from the environment and from experience
8. Acquire capabilities through development
9. Live autonomously within a social community
10. Exhibit self-awareness and a sense of self

These requirements express our common though scientifically informed knowledge about human beings in their habitat. There is no way to know how complete the list is, but many relevant requirements are certainly included.

(1) We list first the requirement to behave flexibly as a function of the environment, since that is the central capability that architectures provide. If a system cannot make itself respond in whatever way is needed, it can hardly be intelligent. The whole purpose of this list, of course, is to go beyond this first item. (2) Flexibility by itself is only a means; it must be in the service of goals and rationally related to obtaining the things and conditions that let the organism survive and propagate. (3) Cognition must operate in real time. This demand of the environment is both important and pervasive. It runs directly counter to the requirements for flexibility, where time to compute is an essential resource. (4) The environment that humans inhabit has important characteristics

1 LINE SHORT
REGULAR
1 LINE LONG

beyond just being dynamic: it is combinatorially rich and detailed, changing simultaneously on many fronts, but with many regularities at every time scale. This affects the cognitive system in several ways. (a) There must be multiple perceptual systems to tap the multiple dynamic aspects; they must all operate concurrently and dynamically, and some must have high bandwidth. (b) There must be very large memories because the environment provides the opportunity to know many relevant things, and in an evolutionary, hence competitive, world opportunity for some produces requirements for all. (c) For the motor system to move around and influence a complex world requires continual determination of many degrees of freedom at a rate dictated by the environment.

(5) Human cognition is able to use symbols and abstractions. (6) It is also able to use language, both natural and artificial. These two requirements might come to the same thing or they might impose somewhat distinct demands. Both are intimately related to the requirement for flexibility and might be redundant with it. But there might be important additional aspects in each. All this need not be settled for the list, which attempts coverage rather than parsimony or independence.

(7) Humans must learn from the environment, not occasionally but continuously and not a little but a lot. This also flows from the multitude of regularities at diverse time scales available to be learned. (8) Furthermore many of our capabilities are acquired through development. When the neonate first arrives, it is surely without many capabilities, but these seem to be exactly the high-level capabilities required to acquire the additional capabilities it needs. Thus there is a chicken-and-egg constraint, which hints at a significant specialization to make development possible. As with the requirements for symbols and language, the relation between learning and development is obscure. Whatever it turns out to be, both requirements belong in the list.

(9) Humans must live autonomously within a social community. This requirement combines two aspects. One aspect of autonomy is greater capability to be free of dependencies on the environment. Relative to the autonomy of current computers and robots, this implies the need for substantially increased capabilities. On the other hand much that we have learned from ethology and social theory speaks to the dependence of individuals on the communities in which they are raised and reside (von Cranach, Foppa, Lepinies, and Ploog 1979). The additional capabilities for low-level autonomy do not negate the extent to which socialization and embedding in a supportive social structure are necessary. If humans leave their communities, they become inept and dysfunctional in many ways. (10) The requirement for self-awareness is somewhat obscure. We surely have a sense of self. But it is not evident what functional role self-awareness plays in the total scheme of mind. Research has made clear the importance from metacognition - considering the capabilities of the self in relation to the task environ-

1 LINE SH
REGU
1 LINE L

ment. But the link from metacognition to the full notion of a sense of self remains obscure.

Human cognition can be taken to be an information-processing system that is a solution to all of the listed requirements plus perhaps others that we have not learned about. Flexibility, the grounds for claiming that human cognition is built on an architecture, is certainly a prominent item, but it is far from the only one. Each of the others plays some role in making human cognition what it is.

The problem of this chapter is *not* what shape cognition as a whole takes in response to these requirements—that is the problem of cognitive science as a whole. Our problem is what is implied by the list for the shape of the architecture. For each requirement there exists a body of general and scientific knowledge, more or less well developed. But cognition is always the resultant of the architecture plus the content of the memories, combined under the impress of being adaptive. This tends to conceal the inner structure and reveal only knowledge-level behavior. Thus extracting the implications for the architecture requires analysis.

Several approaches are possible for such analyses, although we can only touch on them here. The most important one is to get temporally close to the architecture; if there is little time for programmed behavior to act, then the architecture has a chance to shine through. A good example is the exploration of immediate-response behavior that has established an arena of *automatic* behavior, distinguished from an arena of more deliberate *controlled* behavior (Schneider and Shiffrin 1977, Shiffrin and Schneider 1977). Another approach is to look for universal regularities. If some regularity shows through despite all sorts of variation, it may reflect some aspect of the architecture. A good example is the power law of practice, in which the time to perform repeated tasks, almost no matter what the task, improves according to a power law of the number of trials (Newell and Rosenbloom 1981). Architectural mechanisms have been hypothesized to account for it (Rosenbloom and Newell 1986). Yet another approach is to construct experimental architectures that support a number of the requirements in the list. These help to generate candidate mechanisms that will meet various requirements, but also reveal the real nature of the requirement. Many of the efforts in AI and in the development of AI software tools and environments fit this mold (a recent conference (VanLehn 1989) provides a good sampling).

Functional requirements are not the only sources of knowledge about the cognitive architecture. We know the cognitive architecture is realized in neural technology and that it was created by evolution. Both of these have major effects on the architecture. We do not treat either. The implications of the neural structure of the brain are treated in other chapters of this volume, and the implications of evolution, though tantalizing, are difficult to discern.

1 LINE SHORT
REGULAR
1 LINE LONG

3.3 The Nature of the Architecture

We now describe the nature of the cognitive architecture. This is to be given in terms of functions rather than structures and mechanisms. In part this is because the architecture is defined in terms of what it does for cognition. But it is also because, as we have discovered from computer science, an extremely wide variety of structures and mechanisms have proved capable of providing the central functions. Thus no set of structures and mechanisms has emerged as sufficiently necessary to become the criterial features. The purely functional character of architectures is especially important when we move from current digital computers to human cognition. There the underlying system technology (neural circuits) and construction technology (evolution) are very different, so we can expect to see the functions realized in ways quite different from that in current digital technology.

In general the architecture provides support for a given function rather than the entire function. Because an architecture provides a way in which software (that is, content) can guide behavior in flexible ways, essentially all intellectual or control functions can be provided by software. Only in various limiting conditions—of speed, reliability, access to the architectural mechanisms themselves, and the like—is it *necessary* to perform all of the certain functions directly in the architecture. It may, of course, be *efficient* to perform functions in the architecture that could also be provided by software. From either a biological or engineering perspective there is no intrinsic reason to prefer one way of accomplishing a function rather than another. Issues of efficiency, modifiability, constructibility, resource cost, and resource availability join to determine what mechanisms are used to perform a function and how they divide between architectural support, and program and data.

The following list gives known functions of the architecture.

1. Memory
 - a. Contains structures that contain symbol tokens
 - b. Independently modifiable at some grain size
 - c. Sufficient memory
2. Symbols
 - a. Patterns that provide access to distal symbol structures
 - b. A symbol token in the occurrence of a pattern in a structure
 - c. Sufficient symbols
3. Operations
 - a. Processes that take symbol structures as input and produce symbol structures as output
 - b. Complete composability

1 LINE
REC
1 LINE

PER SET
Please Return

4. Interpretation

a. Processes that take symbol structures as input and produce behavior by executing operations

b. Complete interpretability

5. Interaction with the external world

a. Perceptual and motor interfaces

b. Buffering and interrupts

c. Real-time demands for action

c. Continuous acquisition of knowledge

We stress that these functions are only what are known currently. Especially with natural systems such as human cognition, but even with artificial systems, we do not know all the functions that are being performed.² The basic sources of our knowledge of the functions of the architecture is exactly what was skipped over in the previous section, namely the evolution of digital computer architectures and the corresponding abstract theory of machines that has developed in computer science. We do not ground the list in detail in this background, but anyone who wants to work seriously in cognitive architectures should have a fair acquaintance with it (Minsky 1967, Bell and Newell 1971, Hopcroft and Ullman 1979, Siewiorek, Bell, and Newell 1981, Agrawal 1986, Fernandez and Lang 1986, Gajski, Milutinovic, Siegel, and Furht 1987). We now take up the items of this list.

Symbol Systems

The central function of the architecture is to support a system capable of universal computation. Thus the initial functions in our list are those required to provide this capability. We should be able to generate the list simply by an analysis of existing universal machines. However, there are many varieties of universal systems. Indeed a striking feature of the history of investigation of universal computation has been the creation of many alternative independent formulations of universality, all of which have turned out to be equivalent. Turing machines, Markov algorithms, register machines, recursive functions, Pitts-McCulloch neural nets, Post productions, tag systems, plus all manner of digital computer organizations—all include within them a way of formulating a universal machine. These universal machines are all equivalent in flexibility and can all simulate each other. But like architectures (and for the same reasons) each formulation is a framework unto itself, and they often present a quite specific idiosyncratic design, such as the tape, reading head, and five-tuple instruction format of a Turing machine. Although not without a certain charm (special but very general!), this tends to obscure the general functions that are required. The formulation we choose is the *unibo system* (Newell 1980) which is equivalent

1 LINE SHORT
REGULAR
1 LINE LONG

MASTER SET
Please Return

to all the others. The prominent role it gives to symbols has proved useful in discussions of human cognition, however, and its avoidance of specific details of operation makes it less idiosyncratic than other formulations.

The first four items of the list of functions provide the capability for being a symbol system: *memory, symbols, operations, and interpretation*. However, none of these functions (not even symbols) is the function of *representation* of the external world. Symbols do provide an internal representation function, but representation of the external world is a function of the computational system as a whole, so that the architecture supports such representation, but does not itself provide it. (See chapter 2 for how this is possible, and how one moves from the knowledge level, which is *about* the external world, to the symbol level, which contains the mechanisms that provide *aboutness*.)

Memory and Memory Structures The first requirement is for *memory*, which is to say, structures that persist over time. In computers there is a memory hierarchy ranging from working registers within the central processor (such as the address register) to registers used for temporary state (such as an accumulator or operand stack), to primary memory (which is randomly addressed and holds active programs and data), to secondary memory (disks), to tertiary memory (magnetic tapes). This hierarchy is characterized by time constants (speed of access, speed of writing, and expected residency time) and memory capacity, in inverse relation—the slower the memory the more of it is available. The faster memory is an integral part of the operational dynamics of the system and is to be considered in conjunction with it. The larger-capacity, longer-term memory satisfies the requirement for the large amounts of memory needed for human cognition.

Memory is composed of structures, called *symbol structures* because they contain *symbol tokens*. In computers all of the memories hold the same kinds of structures, namely, vectors of bits (bytes and words), although occasionally larger multiples of such units occur (blocks and records). At some sufficiently large grain size the memory structures must be independently modifiable. There are two reasons for this. First, the variety of the external world is combinatorial—it comprises many independent multivalued dimensions located (and iterated) throughout space and time. Only a combinatorial memory structure can hold information about such a world. Second, built-in dependencies in the memory structure, while facilitating certain computations, must ultimately interfere with the ability of the system to compute according to the dictates of the environment. Dependencies in the memory, being unresponsive to dependencies in the environment, then become a drag, even though it may be possible to compensate by additional computation. Within some limits (here called the grain size) of course structures may exhibit various dependencies, which may be useful.

1 LINE SH
REGU
1 LINE L

105 SET
105-1000

Symbols and Symbol Tokens Symbol tokens are patterns in symbol structures that provide access to distal memory structures, that is, to structures elsewhere in memory.⁵ In standard computer architectures a symbol is a memory address and a symbol token is a particular string of bits in a particular word that can be used as an address (by being shipped to the memory-address register, as in figure 3.1). The need for symbols⁶ arises because it is not possible for all of the structure involved in a computation to be assembled ahead of time at the physical site of the computation. Thus it is necessary to travel out to other (distal) parts of the memory to obtain the additional structure. In terms of the knowledge level this is what is required to bring all of the system's knowledge to bear on achieving a goal. It is not possible generally to know in advance all the knowledge that will be used in a computation (for that would imply that the computation had already been carried out). Thus the ingredients for a symbol mechanism are some pattern within the structures being processed (the token), which can be used to open an access path to a distal structure (and which may involve search of the memory) and a retrieval path by means of which the distal structure can be communicated to inform the local site of the computation.⁷

Operations The system is capable of performing operations on symbol structures to compose new symbol structures. There are many variations on such operations in terms of what they do in building new structures or modifying old structures and in terms of how they depend on other symbol structures. The form such operations take in standard computers is the application of an operator to a set of operands, as specified in a fixed instruction format (see figure 3.1). Higher-level programming languages generalize this to the full scope of an applicative formalism, where (Fx_1, x_2, \dots, x_n) commands the system to apply the function F to operands x_1, \dots, x_n to produce a new structure.

Interpretation Some structures (not all) have the property of determining that a sequence of symbol operations occurs on specific symbol structures. These structures are called variously codes, programs, procedures, routines, or plans. The process of applying the operations is called interpreting the symbol structure. In standard computers this occurs by the fetch-execute cycle (compare figure 3.1), whereby each instruction is accessed, its operands decoded and distributed to various registers, and the operation executed. The simplicity of this scheme corresponds to the simplicity of machine language and is dictated by the complexity of what can be efficiently and reliably realized directly in hardware. More complex procedural (high-level) languages can be compiled into an elaborate program in the simpler machine language or executed on the fly (that is, interpretively) by the microcode of a simple subcomputer. Other alternatives are possible, for example, constructing a specific special-purpose machine that embodies the operations of the

1 LINE SHORT
REGULAR
1 LINE LONG

program and then activating the machine. All these come to the same thing—the ability to convert from symbol structures to behavior.

POSTER SET
due Return

The Integrated System We now have all the ingredients of a symbol system. These are sufficient to produce indefinitely flexible behavior (requirement 1 in the first list). Figure 3.2 gives the essential interaction. Operations can construct symbol structures that can be interpreted to specify further operations to construct yet further symbol structures. This loop provides for the construction of arbitrary behavior as a function of other demands. The only additional requirements are some properties of sufficiency and completeness. Without sufficient memory and sufficient symbols the system will be unable to do tasks demanding sufficiently voluminous intermediate references and data, just because it will run out of resources. Without completeness in the loop some sequences of behavior will not be producible. This has two faces: complete composability, so operators can construct any symbol structure, and complete interpretability, so interpretable symbols structures are possible for any arrangement of operations. Under completeness should also be included reliability—if the mechanisms do not operate as posited (including memory), then results need not follow.

(S/P)

Universality is a simple and elegant way to state what it takes to be flexible, by taking flexibility to its limit.⁸ Failures of sufficiency and completeness do not necessarily threaten flexibility in critical ways. In a finite world all resources are finite, but so is the organism's sojourn on earth. Failures of completeness are a little harder to assess because their satisfaction can be extremely devious and indirect (including the uses of error-detecting and correcting mechanisms to deal with finite reliability). But in general there is a continuum of effect in real terms with the severity and extent of the failure. However, no theory is available to inform us about approximations to universality.

In addition to providing flexibility, symbol systems provide important support for several of the other requirements in the first list. For adapt-

Size: 30p6 × 11p6

Figure 3.2 The basic concept of interpretation and construction

106

Figure 3.2

ability (requirement 2) they provide the ability to represent goals and to conditionalize action off of them. For using vast amounts of knowledge (requirement 4.b), they provide symbol structures in which the knowledge can be encoded and arbitrarily large memories with the accompanying ability to access distal knowledge as necessary. For symbols, abstractions, and language (requirements 5 and 6) they provide the ability to manipulate representations. For learning (requirement 7) they provide the ability to create long-term symbol structures.

MASTER SET
Please Return

Interaction with the External World

Symbol systems are components of a larger embedding system that lives in a real dynamic world, and their overall function is to create appropriate interactions of this larger system with that world. The interfaces of the large system to the world are sensory and motor devices. Exactly where it makes sense to say the architecture ends and distinct input/output subsystems begin depends on the particular system. All information processing right up to the energy transducers at the skin might be constructed on a common design and be part of a single architecture, multiple peripheral architectures of distinct design might exist, or multiple specialized systems for transduction and communication might exist that are not architectures according to our definition. Despite all this variability several common functions can be identified: The first is relatively obvious—the architecture must provide for the interfaces that connect the sensory and motor devices to the symbol system. Just what these interfaces do and where they are located is a function of how the boundary is drawn between the central cognitive system (the symbol system) and the peripheral systems.

The second arises from the asynchrony between the internal and external worlds. Symbol systems are an interior milieu protected from the external world, in which information processing in the service of the organism can proceed. One implication is that the external world and the internal symbolic world proceed asynchronously. Thus there must be *buffering* of information between the two in both directions. How many buffer memories and of what characteristics depends on the time constants and rates of the multiple inputs and outputs. If transducers are much slower than internal processing, of course the transducer itself becomes a sufficiently accurate memory. In addition there must be *interrupt* mechanisms to cope with the transfer of processing between the multiple asynchronous sources of information.

The third function arises from the real-time demand characteristics of the external world (requirement 3 in the first list). The environment provides a continually changing kaleidoscope of opportunities and threats, with their own time constants. One implication for the architecture is for an interrupt capability, so that processing can be switched in time to new demands. The mechanics of interruption has already been posited, but the real-time demand also makes clear a requirement

1 LINE SHORT
REGULAR
1 LINE LONG

MASTER SET
Please Return

for precognitive assessment, that is, for assessment that occurs before assessment by the cognitive system. A demand that is more difficult to specify sharply, but is nonetheless real, is that processing be oriented toward getting answers rapidly. This cannot be an unconditional demand, if we take the time constants of the implementation technology as fixed (neural circuits for human cognition), because computing some things faster implies computing other things slower, and more generally there are intrinsic computational complexities. Still, architectures that provide effective time-limited computation are indicated.

The fourth function arises from an implication of a changing environment—the system cannot know in advance everything it needs to know about such an environment. Therefore the system must continually acquire knowledge from the environment (part of requirement 7) and do so at time constants dictated by the environment (a less obvious form of requirement 3). Symbol systems have the capability of acquiring knowledge, so in this respect at least no new architectural function is involved. The knowledge to be acquired flows in from the environment in real time, however, and not under the control of the system. It follows that learning must also occur essentially in real time. In part this is just the dynamics of the bathtub—on average the inflow to a bathtub (here encoded experience) must equal the outflow from the bathtub (here experience processed to become knowledge). But it is coupled with the fact that the water never stops flowing in, so that there is no opportunity to process at leisure.

Summary

We have attempted to list the functions of the cognitive architecture, which is to provide the support for human cognition, as characterized in the list of requirements for shaping the architecture. Together the symbol-system and real-time functions cover a large part of the primitive functionality needed for requirements 1 through 7. They do not ensure that the requirements are met, but they do provide needed support.

There is little to say for now about architectural support for development (requirement 8). The difficulty is our minimal knowledge of the mechanisms involved in enabling developmental transitions, even at a psychological level (Klahr 1989). It makes a significant difference whether development occurs through the type of general learning that is supported by symbol systems—that is, the creation of long-term symbol structures—or by distinct mechanisms. Even if development were a part of general learning after the first several years, it might require special architectural mechanisms at the beginning of life. Such requirements might shape the architecture in many other ways.

Autonomy in a social environment is another requirement (number 9 in the list) where we cannot yet pin down additional functions for the architecture to support. On the more general issue of autonomy, however, issues that have proved important in computer architecture

1 LINE
REC
1 LINE

MASTER SET
Please Return

include protection, resource allocation, and exception handling. Protection enables multiple components of a system to behave concurrently without interfering with each other. Resource allocation enables a system to work within its finite resources of time and memory, recycling resources as they are freed. Exception handling enables a system to recover from error situations that would otherwise require intervention by a programmer (for example, a division by zero or the detection of an inconsistency in a logic data base).

Issues of self-awareness (requirement 10) have recently been an active research topic in computer science, under the banner of metalevel architecture and reflection (the articles in Maes and Nardi 1988 provide a good sampling). Functionalities studied include how a system can model, reason about, control, and modify itself. Techniques for exception handling turn out to be a special case of the ability of a system to reason about and modify itself. On the psychological side the work on metacognition has made us aware of the way knowledge of a person's own capabilities (or lack of) affects performance (Brown 1978). As yet this work does not seem to have clear implications for the architecture, because it is focused on the development and use of adaptive strategies that do not seem to require special access to the instantaneous running state of the system, which is the obvious architectural support issue.

3.4 Example Architectures: Act* and Soar

We now have an analysis of the functions of a cognitive architecture and the general way it responds to the requirements of our first list. To make this analysis concrete, we examine two cognitive architectures, Act* (Anderson 1983) and Soar (Laird, Newell, and Rosenbloom 1987). Act* is the first theory of the cognitive architecture with sufficient detail and completeness to be worthy of the name. It represents a long development (Anderson and Bower 1973, Anderson 1976), and further developments have occurred since the definitive book was written (Anderson 1986, Anderson and Thompson 1988). Soar is a more recent entry as a cognitive theory (Newell 1987, Polk and Newell 1988, Rosenbloom, Laird, and Newell 1988). Its immediate prior history is as an AI architecture (Laird, Rosenbloom, and Newell 1986, Steir et al. 1987), but it has roots in earlier psychological work (Newell and Simon 1972, Newell 1973, Rosenbloom and Newell 1988). Using two architectures provides some variety to help clarify points and also permits a certain amount of comparison. Our purpose is to make clear the nature of the cognitive architecture, however, rather than to produce a judgment between the architectures.

Overview

Let us start with a quick overview of the two systems and then proceed to iterate through the functions in the second list. Figure 3.3 gives the

1 LINE SHORT
REGULAR
1 LINE LONG

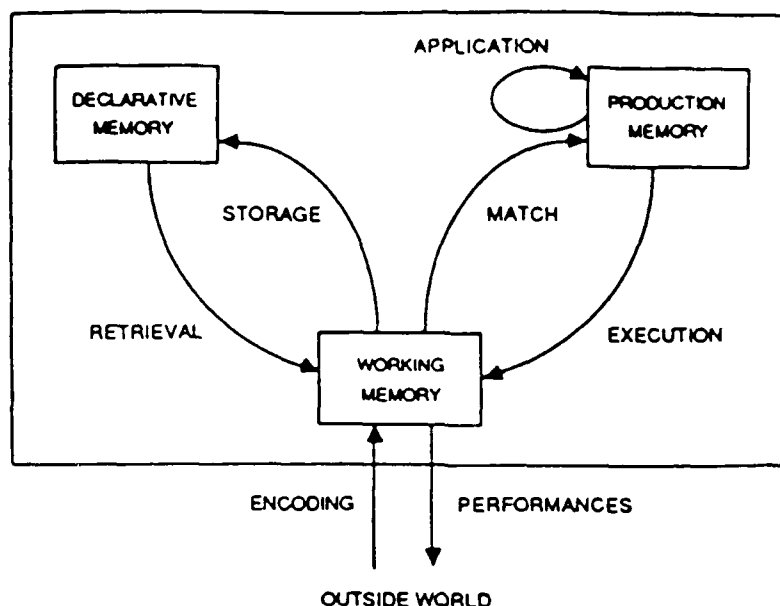


Figure 3.3 Overview of the Act* cognitive architecture (Anderson 1983)

basic structure of Act*. There is a long-term declarative memory in the form of a semantic net. There is a long-term procedural memory in the form of productions. Strengths are associated with each long-term memory element (both network nodes and productions) as a function of its use. Each production has a set of conditions that test elements of a working memory and a set of actions that create new structures in the working memory. The working memory is activation based; it contains the activated portion of the declarative memory plus declarative structures generated by production firings and perception.⁴ Activation spreads automatically (as a function of node strength) through working memory and from there to other connected nodes in the declarative memory. Working memory may contain goals that serve as large sources of activation. Activation, along with production strength, determines how fast the matching of productions proceeds. Selection of productions to fire is a competitive process between productions matching the same data. New productions are created by compiling the effects of a sequence of production firings and retrievals from declarative memory so that the new productions can move directly from initial situations to final results.¹⁰ Whenever a new element is created in working memory there is a fixed probability it will be stored in declarative memory.

Figure 3.4 provides a corresponding overview of Soar. There is a single long-term memory—a production system—that is used for both declarative and procedural knowledge. There is a working memory that contains a goal hierarchy, information associated with the goal hierarchy, preferences about what should be done, perceptual information, and motor commands. Interaction with the outside world occurs via

1 LINE SH
REGU
1 LINE LO

Size: 27p0×24p0

Figure 3.4 Overview of the Soar cognitive architecture

interfaces between working memory and one or more perceptual and motor systems. All tasks are formulated as searches in problem spaces, that is, as starting from some initial state in a space and finding a desired state by the application of the operators that comprise the space. Instead of making decisions about what productions to execute—all productions that successfully match are fired in parallel—decisions are made about what problem spaces, states, and operators to utilize. These decisions are based on preferences retrieved from production memory into working memory. When a decision proves problematic (because of incomplete or inconsistent knowledge), a subgoal is automatically created by the architecture and problem solving recurses on the task of resolving the impasse in decision making. This generates a hierarchy of goals and thus problem spaces. New productions are created continuously from the traces of Soar's experience in goal-based problem solving (a process called chunking).

Memory

Memory is to be identified by asking what persists over time that can be created and modified by the system. Both Act* and Soar have memory hierarchies that range in both time constants and volume. At the small, rapid end both have working memories. Working memory is a

1 LINE SHORT
REGULAR
1 LINE LONG

MASTER SET
Please Return

temporary memory that cannot hold data over any extended duration. In Act* this is manifest because working memory is an activated subset of declarative memory and thus ebbs and flows with the processing of activation. In Soar working memory appears as a distinct memory. Its short-term character derives from its being linked with goals and their problem spaces, so that it disappears automatically as these goals are resolved.

Beyond working memory both architectures have permanent memories of unbounded size, as required for universality. Act* has two such memories—declarative memory and production memory—with strengths associated with each element in each memory. The normal path taken by new knowledge in Act* is from working memory to declarative memory to production memory. Declarative memory comes before production memory in the hierarchy because it has shorter storage and access times (though it cannot lead directly to action). Soar has only one permanent memory of unbounded size—production memory—which is used for both declarative and procedural knowledge. Soar does not utilize strengths.

The above picture is that Act* has two totally distinct memories, and Soar has one that is similar to one of Act*'s memories. However, this conventional surface description conceals some important aspects. One is that Act* and Soar productions do not function in the same way in their respective systems (despite having essentially the same condition-action form). Act* productions correspond to problem-solving operators. This is essentially the way productions are used throughout the AI and expert-systems world. Soar productions operate as an associative memory. The action side of a production contains the symbol structures that are held in the memory; the condition side provides the access path to these symbol structures. Firing a Soar production is the act of retrieval of its symbol structures. Operators in Soar are implemented by collections of productions (or search in subgoals). Another hidden feature is that Act*'s production memory is realized as a network structure similar in many ways to its semantic net. The main effect is that activation governs the rate of matching of productions in the same way that activation spreads through the declarative network. Thus these two memories are not as distinct as it might seem.

In both Act* and Soar the granularity of long-term memory (the independently modifiable unit) is relatively fine, being the individual production and, for Act*'s declarative memory, the node and link. This is a much larger unit than the word in conventional computers (by about two orders of magnitude) but much smaller than the frame or schema (again by about two orders of magnitude). This is an important architectural feature. The frame and schema have been introduced on the hypothesis that the unit of memory organization needs to be relatively large to express the organized character of human thought (Minsky 1975). It is not easy to make size comparisons between units of

1 LINE SHOR
REGULA
1 LINE LONG

memory organization because this is one place where the idiosyncratic world-view character of architectures is most in evidence, and all memory organization have various larger and smaller hierarchical units. Nevertheless both Act* and Soar are on the fine-grained side.

The memory structures of both Act* and Soar are the discrete symbolic structures familiar from systems such as Lisp. There are differences in detail. Soar has a uniform scheme of objects with sets of attributes and values. Act* has several primitive data structures: attributes and values (taken to be the abstract propositional code), strings (taken to be the temporal code), and metric arrays (taken to be the spatial code). The primary requirement of a data structure is combinatorial variability, and all these structures possess it. The secondary considerations are on the operations that are required to read and manipulate the data structures, corresponding to what the structures are to represent. Thus standard computers invariably have multiple data structures, each with associated primitive operations, for example, for arithmetic or text processing. Act* here is taking a clue from this standard practice.

Symbols

Symbols are to be identified by finding the mechanisms that provide distal access to memory structures that are not already involved in processing. For both Act* and Soar this is the pattern match of the production system, which is a process that starts with symbolic structures in working memory and determines that a production anywhere in the long-term memory will fire. The symbol tokens here are the combinations of working memory elements that match production conditions. Each production's left-hand side is a symbol.

For Soar this is the only mechanism for distal access (working memory being essentially local). For Act* there is also a mechanism for distal access to its declarative memory, in fact a combination of two mechanisms. First, each token brought into working memory by firing a production (or by perception) makes contact with its corresponding node in the declarative semantic net. Second, spreading activation then operates to provide access to associated nodes.

It is useful to identify the pair of features that gives symbolic access in production systems its particular flavor. The first feature is the context-dependent nature of the production match. Simple machine addresses act as context-independent symbols. No matter what other structures exist, the address causes information to be retrieved from the same location.¹¹ In a production system a particular pattern can be a symbol that results in context-independent access to memory structures, or (more typically) it can be conjoined with additional context patterns to form a more complex symbol that constrains access to occur only when the context is present.

The second feature is the recognition nature of the production match. Traditional computers access memory either by pointers to arbitrary

1 LINE SHORT
REGULAR
1 LINE LONG

memory locations (random access) or by sequential access to adjacent locations (large secondary stores). In production systems symbols are constructed out of the same material that is being processed for the task, so memory access has a *recognition, associative, or content-addressed* nature. All schemes can support universality; however, the recognition scheme is responsive to two additional cognitive requirements. First, (approximately) constant-time access to the whole of memory is responsive to the *real-time requirement*. This includes random-access and recognition memories, but excludes sequential access systems such as Turing machines. But specific task-relevant accessing schemes must be constructed, or the system is doomed to operate by generate and test (and might as well be a tape machine). Recognition memories construct the accessing paths from the ingredients of the task and thus avoid deliberate acts of construction, which are required by location-pointer schemes. This may actually be an essential requirement for a learning system that has to develop entirely on its own. Standard programming involves intelligent programmers who invent specific accessing schemes based on deep analysis of a task.¹²

Operations

The operations are to be identified by asking how new structures get built and established in the long-term memory. In standard computer systems the form in which the operations are given is dictated by the needs of interpretation, that is, by the structure of the programming language. Typically everything is fit within an operation-operand structure, and there is a single, heterogeneous set of all primitive operation codes—load, store, add, subtract, and, or, branch-on-zero, execute, and so on. Some of these are operations that produce new symbol structures in memory, but others affect control or deal with input/output.

Production systems, as classically defined, also operate this way. The right-hand-side actions are operation-operand structures that can specify general procedures, although a standard set of operations are predefined (make, replace, delete, write . . .). In some it is possible to execute a specified production or production system on the right-hand side, thus providing substantial right-hand-side control. But Act* and Soar use a quite different scheme.

The right-hand-side action becomes essentially just the operation of creating structures in working memory. This operation combines focusing, modifying, and creating—it brings existing structures into working memory, it creates working memory structures that relate existing structures, and it creates new structures if they do not exist. This operation is coextensive with the retrieval of knowledge from long-term memory (production firing). The dependence of the operation on existing structures (that is, its inputs) occurs by the matching of the production conditions. It is this matching against what is already in working memory that permits the multiple functions of focusing, modifying, and

1 LINE SH
REGU
1 LINE L

WATER SET
PAGE 115

creating to be distinguished and to occur in the appropriate circumstances, automatically, so to speak. Along with this the act of retrieval from long-term memory (into working memory) does not happen as a distinct operation that reproduces the content of long-term memory in working memory. Rather each retrieval is an act of computation (indeed computation takes place only in concert with such retrievals), so that working memory is never the same as stored memory and is always to some extent an adaptation of the past to the present.

In Act* and Soar storing information in long-term memory is separated from the act of computation in working memory. It is incorporated as learning new productions, called *production compilation* in Act* and *chunking* in Soar, but similar operations nonetheless. The context of production acquisition is the occasion of goal satisfaction or termination, and the constructed production spans from the conditions holding before the goal to the actions that caused the final resolution. The production is simply added to the long-term production memory and becomes indistinguishable from any other production. Such a production is functional, producing in one step what took many steps originally. It also constitutes an implicit form of generalization in that its conditions are extracted from the total context of working memory at learning time, and so can be evoked in situations that can be arbitrarily different in ways that are irrelevant to these conditions. Production compilation and chunking go considerably beyond the minimal support for experiential learning provided by a standard symbol system. Without deliberate effort or choice they automatically acquire new knowledge that is a function of their experiences.

Act* has other forms of memory aside from the productions and necessarily must have operations for storing in each of them. They are all automatic operations that do not occur under deliberate control of the system. One is the strength of productions, which governs how fast they are processed and hence whether they become active in an actual situation. Every successful firing of a production raises its strength a little and hence increases the likelihood that if satisfied it will actually fire (another form of experiential learning). The second is storage in declarative memory. Here there is simply a constant probability that a newly created element will become a permanent part of declarative memory. Declarative learning is responsive to the requirement of learning from the environment. In Soar chunking performs this function in addition to its function of learning from experience.

Interpretation

Interpretation is to be identified by finding where a system makes its behavior dependent on the symbolic structures in its long-term memory, in particular, on structures that it itself created earlier. A seemingly equivalent way is to find what memory structures correspond to the program in typical computer systems, namely, the symbol structures

1 LINE SHORT
REGULAR
1 LINE LONG

MASTER SET
Please Return

that specify a sequence of operations: do this, then do that, then do this, although also admitting conditionals and calls to subprocedures. Namely, one seeks compact symbol structures that control behavior over an extended interval.

One looks in vain for such symbol structures in the basic descriptions of the Act* and Soar architectures. (Program structures can exist of course, but they require software interpreters.) Memory-dependent behavior clearly occurs, however, and is derived from multiple sources—production systems, problem-solving control knowledge, and goal structures.

The first source in both systems is the form of interpretation inherent in production systems. A production system shreds control out into independent fragments (the individual productions) spread out all over production memory, with data elements in working memory entering in at every cycle. This control regime is often referred to as *data directed*, in contrast to *goal directed*, but this characterization misses some important aspects. Another way to look at it is as a *recognize-act cycle* in contrast to the classical *fetch-execute cycle* that characterizes standard computers. According to this view, an important dimension of interpretation is the amount of decision making that goes on between steps. The fetch-execute cycle essentially has only a pointer into a plan and has to take deliberate steps (doing tests and branches) to obtain any conditionality at all. The recognize-act cycle opens up the interpretation at every point to anything that the present working memory can suggest. This puts the production match *inside* the interpretation cycle.

The second source is the control knowledge used to select problem-solving operators. In Act* the productions are the problem-solving operators. As described in the previous paragraph, production selection is a function of the match between working memory elements and production conditions. Several additional factors, however, also come into play in determining the rate of matching and thus whether a production is selected for execution. One factor is the activation of the working memory elements being matched. A second factor is the strength of the production being matched. A third factor is the competition between productions that match the same working memory elements in different ways.

In Soar problem-solving operators are selected through a two-phase *decision cycle*. First, during the elaboration phase the long-term production memory is accessed repeatedly—initial retrievals may evoke additional retrievals—in parallel (there is no conflict resolution) until quiescence. Any elements can be retrieved, but among these are *preferences* that state which of the operators are acceptable, rejectable, or preferable to others. When all the information possible has been accumulated, the decision procedure winnows the available preferences and makes the next decision, which then moves the system to the next cycle.

1 LINE SHOR
REGULA
1 LINE LONG

In fact Soar uses this same basic interpreter for more than just selecting what operator to execute. It is always trying to make the decisions required to operate in a problem space: to decide what problem space to use, what state to use in that problem space, what operator to use at that state, and what state to use as the result of the operator. This is what forces all activity to take place in problem spaces. This contrasts with the standard computer, which assumes all activity occurs by following an arbitrary program.

The third source of memory-dependent behavior is the use of goal structures. Act* provides special architectural support for a goal hierarchy in its working memory. The current goal is a high source of activation, which therefore operates to focus attention by giving prominence to productions that have it as one of their conditions. The architecture takes care of the tasks of popping successful subgoals and moving the focus to subsequent subgoals, providing a depth-first traversal of the goal hierarchy. Thus the characterization of data- versus goal-directed processing is somewhat wide of the mark. Act* is a paradigm example of an AI system that uses goals and methods to achieve adaptability (requirement 2 in the first list). Complex tasks are controlled by productions that build up the goal hierarchy by adding conjunctions of goals to be achieved in the future.

Soar uses a much less deliberate strategy for the generation of goals. When the decision procedure cannot produce a single decision from the collection of preferences that happen to accumulate—because, for example, no options remain acceptable or several indistinguishable options remain—an impasse is reached. Soar assumes this indicates lack of knowledge—given additional knowledge of its preferences, a decision could have been reached. It thus creates a subgoal to resolve this impasse. An impasse is resolved just when preferences are generated, of whatever nature, that lead to a decision at the higher level. Thus Soar generates its own subgoals out of the impasses the architecture can detect, in contrast to Act*, which generates its subgoals by deliberate action on the part of its productions. The effect of deliberate subgoals is achieved in Soar by the combination of an operator, which is deliberately generated and selected, and an impasse that occurs if productions do not exist that implement the operator. In the subgoal for this impasse the operator acts as the specification of a goal to be achieved.

Interaction with the External World

Act*, as is typical of many theories of cognition, focuses on the central architecture. Perception and motor behavior are assumed to take place in additional processing systems off stage. Input arrives in working, which thus acts as a buffer between the unpredictable stream of environmental events and the cognitive system. Beyond this, however, the architecture has simply not been elaborated on in these directions.

Soar has the beginnings of a complete architecture, which embeds

1 LINE SHORT
REGULAR
1 LINE LONG

the central architecture within a structure for interacting with the external world. As shown in figure 3.4, Soar is taken as a controller of a dynamic system interacting with a dynamic external environment (located across the bottom of the figure). There are processes that transduce the energies in the environment into signals for the system. Collectively they are called *perception*, although they are tied down only on the sensory side (transduction from the environment). Similarly there are processes that affect the environment. Collectively they are called the *motor system*, although they are tied down only on the physical action side. As in Act* working memory serves as the buffer between the environment and central cognition.

The total system consists of more than perception to central cognition to the motor system. There are productions, called *encoding productions* and *decoding productions*. These are identical in form and structure to the productions of central cognition. They differ only in being independent of the decision cycle—they just run free. On the input side, as elements arrive autonomously from the perceptual system, encoding productions provide what could be termed perceptual parsing, putting the elements into a form to be considered by central cognition. On the output side decoding productions provide what could be called motor-program decoding of commands produced by the cognitive system into the form used by the motor system. The motor system itself may produce elements back into the working memory (possibly parsed by encoding productions), permitting monitoring and adjustment.

All this activity is not under control; these productions recognize and execute at will, concurrently with each other and central cognition. Control is exercised by central cognition, which can now be seen to consist essentially of just the architecture of the decision mechanism, from which flows the decision cycle, impasses, the goal stack, and the problem-space organization. Further, central cognition operates essentially as a form of localized supervisory control over autonomous and continuing activities in working memory generated by the perception systems, the motor systems, and their coupled encoding and decoding productions.

This permits an understanding of an architectural question that has consumed a lot of attention, namely, wherein lies the serial character of cognition? Central cognition is indeed serial, which is what the decision mechanism enforces, and so it can consider only some of what goes on in the working memory. The serial system is imposed on a sea of autonomous parallel activity to attain control, that is, for the system to be able to prevent from occurring actions that are not in its interests. Thus seriality is a designed feature of the system. Seriality can occur for other reasons as well, which can be summarized generally as resource constraints or bottlenecks. Such bottlenecks can arise from the nature of the underlying technology and thus be a limitation on the system.

Interrupt capabilities are to be identified by finding where behavior can switch from one course to another that is radically different. For Act* switching occurs by the basic maximum-selecting property of an activation mechanism—whatever process can manage the highest activation can take over the control of behavior. For Soar switching occurs by the decision cycle—whatever can marshal the right preferences compared with competing alternatives can control behavior. Switching can thus occur at a fine grain. For both Act* and Soar their basic switching mechanism is also an interrupt mechanism, because alternatives from throughout the system compete on an equal basis. This arises from the open character of production systems that contact all of memory at each cycle. Thus radical changes of course can occur at any instant. This contrasts with standard computers. Although arbitrary switching is possible at each instruction (for example, branch on zero to an arbitrary program), such shifts must be determined deliberately and by the (pre-constructed) program that already has control. Thus the issue for the standard computer is how to be interrupted, whereas the issue for Soar and Act* (and presumably for human cognition) is how to keep focused.

Learning from the environment involves the long-term storage of structures that are based on inputs to the system. Act* stores new inputs into declarative memory with a fixed probability, from which inputs can get into production memory via compilation, a process that should be able to keep pace with the demands of a changing environment. Soar stores new inputs in production memory via chunking. This implies that an input must be used in a subgoal for it to be stored, and that the bandwidth from the environment into long-term memory will be a function of the rate at which environmental inputs can be used.

Summary

We have now instantiated the functions of the cognitive architecture for two architectures, Soar and Act*, using their commonalities and differences to make evident how their structures realize these functions. The commonalities of Act* and Soar are appreciable, mostly because both are built around production systems. We have seen that production systems, or more generally, recognition-based architectures, are a species of architecture that is responsive to the real-time requirement, which is clearly one of the most powerful shapers of the architecture beyond the basic need for symbolic computation.

The move to production systems, however, is only the first of three major steps that have moved Act* and Soar jointly into a very different part of the architecture space from all of the classical computers. The second step is the abandonment of the application formalism of applying operations to operands. This abandonment is not an intrinsic part of production systems, as evidenced by the almost universal use of application on the action side of productions. This second step locks the operations on symbolic structures into the acts of memory retrieval.

1 LINE SHORT
REGULAR
1 LINE LONG

The third step is the separation of the act of storing symbolic structures in long-term memory—the learning mechanisms of Act* and Soar—from the deliberate acts of performing tasks.

There are some architectural differences between Act* and Soar, though not all appear to be major differences when examined carefully. One example is the dual declarative and procedural memory of Act* versus the single production memory of Soar. Another is the use of activation in Act* versus the use of accumulated production executions (the elaboration phase) in Soar. A third is the commitment to multiple problem spaces and the impasse mechanism in Soar versus the single-space environment with deliberate subgoals in Act*. Thus these architectures differ enough to explore a region of the architecture space.

The downside of using two closely related architectures for the exposition is that we fail to convey an appreciation of how varied and rich in alternatives the space of architecture is. With a slight stretch we might contend we have touched three points in the architecture space: classical (von Neumann) architectures, classical production systems, and Act* and Soar. But we could have profitably examined applicative languages (for example, Lisp; see Steele 1984), logic programming languages (for example, Prolog; see Clocksin and Mellish 1984), frame (or schema) systems (for example, KLONE; see Brachman 1979), blackboard architectures (for example, BB1; see Hayes-Roth 1985), and others as well. Also we could have explored the effect of parallelism, which itself has many architectural dimensions. This was excluded because it is primarily driven by the need to exploit or compensate for implementation technology, although (as has been pointed out many times) it can also serve as a response to the real-time requirement.

3.5 The Uses of the Architecture

Given that the architecture is a component of the human cognitive system, it requires no justification to spend scientific effort on it. Understanding the architecture is a scientific project in its own right. The architecture, however, as the frame in terms of which all processing is done and the locus of the structural constraints on human cognition, would appear to be the central element in a general theory of cognition. This would seem to imply that the architecture enters into all aspects of cognition. What keeps this implication at bay is the fact (oft noted, by now) that architectures hide themselves beneath the knowledge level. For many aspects of human cognitive life, what counts are the goals, the task situation, and the background knowledge (including education, socialization, and enculturation). So the architecture may be critical for cognition, just as biochemistry is, but with only circumscribed consequences for ongoing behavior and its study.

How then is a detailed theory of the architecture to be used in cognitive science, other than simply filling in its own part of the picture?

1 LINE SH
REGU
1 LINE L

There are four partial answers to this question, which we take up in turn.

Establishing Gross Parameters

The first answer presupposes that the architecture has large effects on cognition, but that these effects can be summarized in a small set of gross parameters. The following list assembles one set of such parameters, which are familiar to all cognitive scientists—the size of short-term memory, the time for an elementary operation, the time to make a move in a problem space, and the rate of acquisition into long-term memory:

1. Memory Unit: 1 chunk composed of 3 subchunks
2. Short-term memory size: 3 chunks plus 4 chunks from long-term memory
3. Time per elementary operation: 100 ms
4. Time per step in a problem space: 2 s
5. Time to learn new material: 1 chunk per 2 s

It is not possible to reason from parameters alone. Parameters always imply a background model. Even to fill in the list it is necessary to define a unit of memory (the chunk), which then already implies a hierarchical memory structure. Likewise to put a sequence of elementary operations together and sum their times is already to define a functionally serial processing structure.

The block diagrams that have been a standard feature in cognitive psychology since the mid-1950s (Broadbent 1954) express the sort of minimal architectural structure involved. Mostly they are too sketchy, in particular, providing only a picture of the memories and their transfer paths. A somewhat more complete version, called the *model human processor* (Card, Moran, and Newell 1983), is shown in figure 3.5, which indicates not only the memories but a processor structure of three parallel processors—perceptual, cognitive, and motor. Perception and motor systems involve multiple concurrent processors for different modalities and muscle systems, but there is only a single cognitive processor. The parameters of the two figures have much in common. Figure 3.5 is of course a static picture. By its very lack of detail it implies the simplest of processing structures. In fact it can be supplemented by some moderately explicit general principles of operations (Card, Moran, and Newell 1983), such as that uncertainty always increases processing time. These principles provide some help in using it, but are a long way from making the scheme into a complete architecture. In particular the elementary operations and the details of interpretation remain essentially undefined.

If the way the architecture influences behavior can be summarized by a small set of parameters plus a simple abstract background model,

1 LINE SHORT
REGULAR
1 LINE LONG

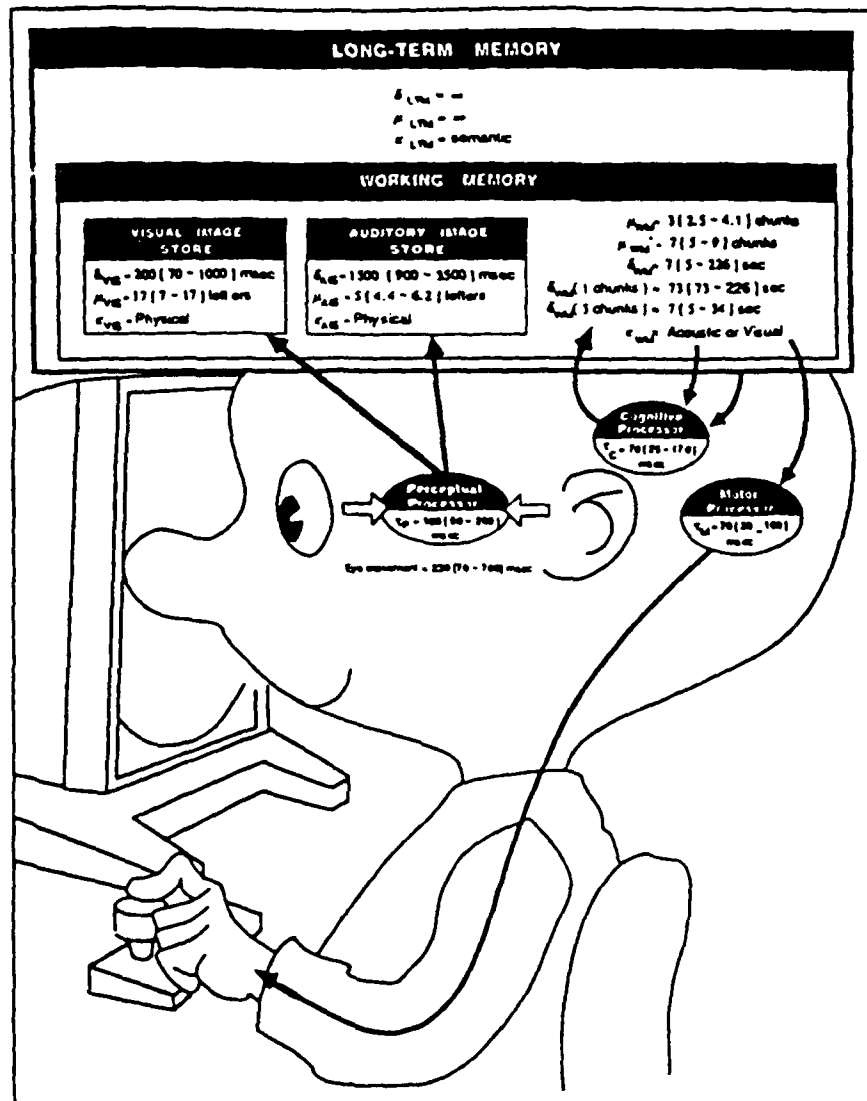


Figure 3.5 The model human processor block diagram (Card, Moran, and Newell 1983)

then the contribution from studying the architecture is twofold. First, given an established scheme such as figure 3.5, the parameters need to be pinned down, their variability understood, their mutability discovered, their limits of use assessed, and so on. Second, the gross processing model can be wrong, not in being a gross approximation (which is presumed) but in being the wrong sort of formulation. If it were replaced with a quite different formulation, then inferences might become easier, a broader set of inferences might be possible, and so forth. One such example from the mid-1970s is the replacement of the multi-store model of memory with a model containing only a single memory within which information is distinguished only by the depth to which it has been processed (Craik and Lockhart 1972).

The Form of Simple Cognitive Behavior

To perform a complex task involves doing a sequence of basic operations in an arrangement conditional on the input data. Much of psychological interest depends on knowing the sequence of operations that humans perform for a given task, and much psychological effort, both experimental and theoretical, is focused on finding such sequences. This is especially true for tasks that are primarily cognitive, in which perceptual and motor operations play only a small role in the total sequence.

The architecture dictates both the basic operations and the form in which arrangements of operations are specified, namely, the way behavior specifications are encoded symbolically. Thus the architecture plays some role in determining such sequences. For tasks of any complexity, however, successful behavior can be realized in many different ways. At bottom this is simply the brute fact that different methods or algorithms exist for a given task, as when a space can be searched depth first, breadth first, or best first, or a column of numbers can be added from the top or the bottom. Thus writing down the sequence of operations followed by a subject for a complex task, given just the architecture and the task, is nearly impossible. It is far too underdetermined, and the other factors, which we summarize as the subject's knowledge, are all important.

As the time to perform the task gets shorter, however, the options get fewer for what sequences could perform a task. Indeed suppose the time constant of the primitive data operations of the architecture is about 100 ms, and we ask for a task to be performed in about 0.1 ms, then the answer is clear without further ado: it cannot be done. It makes no difference how simple the task is (for example, are two names the same?). Suppose the task is to be performed in about 100 ms. Then a scan of the architecture's basic processes will reveal what can be done in a single operation time. If the performance is possible at all, it is likely to be unique—there is only one way to test for name equality in a single basic operation, though conceivably an architecture might offer some small finite number of alternatives. As the time available grows,

1 LINE SHORT
REGULAR
1 LINE LONG

MASTER SET
Please Return

what can be accomplished grows and the number of ways of accomplishing a given task grow. If 100 s is available, then there are probably several ways to determine whether two names are the same. The constraint comes back, however, if the demands of the performance grow apace.

Thus there is a region in which knowing the architecture makes possible plausible guesses about what operation sequences humans will use on a task. Consider the memory-scanning task explored by Sternberg and discussed by Bower and Clapper in chapter 7 in connection with the additive-factors methodology. The subject first sees a set of items H, P, Z and then a probe item Q and must say as quickly as possible whether the probe was one of the sequence. Three regularities made this experiment famous. First, the time to answer is linear in the size of the test set (response time = $400 + 3 \cdot 40 = 520$ ms for the case above), strongly suggesting the use of a serial scan (and test) at 40 ms per item. Second, the same linear relation with the same slope of 40 ms per item holds whether the probe is or is not in the list. This contradicts the obvious strategy of terminating the scan when an item is found that matches the probe, which would lead to an apparent average search rate for positive probes that is half as large as that for negative probes—on average only half of the list would be scanned for a positive probe before the item is found. Third, the scan rate (40 ms per item) is *very* fast—humans take more than 100 ms per letter to say the alphabet to themselves.

As Bower and Clapper (chapter 7) report, this experimental situation has been explored in many different ways and has given rise to an important experimental method (*additive factors*) to assess how different factors entered into the phenomena. For us the focus is on the speed with which things seem to happen. Speeded reaction times of about 400 ms already get close to the architecture, and phenomena that happen an order of magnitude faster (40 ms per item) must be getting close to the architecture floor. This is especially true when one considers that neurons are essentially 1-ms devices, so that neural circuits are 10-ms devices.

Just given the Sternberg phenomena, including these tight bounds, one cannot infer the mechanisms that perform it. In fact the Sternberg situation has been studied to show that one cannot even infer whether the “search” is going on in series or in parallel. Given an architecture, however, the picture becomes quite different. For instance, given Act*, the time constants imply that productions take a relatively long time to fire, on the order of 100 ms. Thus the Sternberg effect cannot be due to multiple production firings. Hence it must be a spreading activation phenomenon. *Indeed the explanation that Anderson offers for the Sternberg effect is based on spreading activation* (Anderson 1983, pp. 119–120). Two productions, one to say yes if the probe is there and one to say no if the probe is not, define the subject’s way of doing the task.

1 LINE SHOR
REGULA
1 LINE LONG

and then a calculation based on the flow of activation shows that it approximates the effect. The important point for us is that the two productions are the obvious way to specify the task in Act*, and there are few if any alternatives.

If we turn to Soar, there is an analogous analysis. First, general timing constraints imply that productions must be 10-ms mechanisms, so that the decision cycle is essentially a 100-ms mechanism, although speeded tasks would force it down (Newell 1987). Thus selecting and executing operators will take too long to be used to search and process the items of the set (at 40 ms each). Thus the Sternberg effect must be occurring within a single decision cycle, and processing the set must occur by executing a small number of productions (one to three) on each item. The fact that the decision cycle runs to quiescence would seem to be related to all items of the set being processed, whether the probe matches or not. These constraints do not pin down the exact program for the memory-scanning task as completely as in Act*, but they do specify many features of it. Again the important point here is that the closer the task is to the architecture, the more the actual program used by humans can be predicted from the structure of the architecture.

Hidden Connections

An architecture provides a form of unification for cognitive science, which arises, as we have seen, from all humans accomplishing all activities by means of the same set of mechanisms. As we have also seen, these common mechanisms work through content (that is, knowledge), which varies over persons, tasks, occasions, and history. Thus there is immense variability in behavior, and many phenomena of cognitive life are due to these other sources.

One important potential role for studies of the architecture is to reveal hidden connections between activities that on the basis of content and situation seem quite distant from each other. The connections arise, of course, because of the grounding in the same mechanisms of the architecture, so that, given the architecture, they may be neither subtle nor obscure. One such example is the way chunking has turned out to play a central role in many different forms of learning—such as the acquisition of macrooperators, the acquisition of search-control heuristics, the acquisition of new knowledge, constraint compilation, learning from external advice, and so on—and even in such traditionally non-learning behaviors as the creation of abstract plans (Steier et al. 1987). Previously, special-purpose mechanisms were developed for these various activities.

In addition to the joy that comes directly from discovering the cause of any scientific regularity, revealing distal connections is useful in adding to the constraint that is available in discovering the explanation for phenomena. An example—again from the domain of Soar—is how chunking, whose route into Soar was via a model of human practice,

ET?

1 LINE SHORT
REGULAR
1 LINE LONG

has provided the beginnings of a highly constrained model of verbal learning (Rosenbloom, Laird, and Newell 1988). Using chunking as the basis for verbal learning forces it to proceed in a reconstructive fashion—learning to recall a presented item requires the construction of an internal representation of the item from structures that can already be recalled—and is driving the model, for functional reasons, to where it has many similarities to the EPAM model of verbal learning (Feigenbaum and Simon 1984).

Removing Theoretical Degrees of Freedom

One of the itches of cognitive scientists ever since the early days of computer simulation of cognition is that to get a simulation to run, it is necessary to specify many procedures and data structures that have no psychological justification. There is nothing in the structure of the simulation program that indicates which procedures (or more generally which aspects) make psychological claims and which do not.

One small but real result of complete architectures is to provide relief for this itch. A proposal for an architecture is a proposal for a complete operational system. No additional processes are required or indeed even possible. Thus when a simulation occurs within such an architecture, all aspects of the system represent empirical claims. This can be seen in the case of Soar—the decision cycle is claimed to be how humans make choices about what to do, and impasses are claimed to be real and to lead to chunking. Each production is claimed to be psychologically real and to correspond to an accessible bit of knowledge of the human. And the claims go on. A similar set of claims can be made for Act*. Many of these claims (for Act* or Soar) can be, and indeed no doubt are, false. That is simply the fate of inadequate and wrong theories that fail to correspond to reality. But there is no special status of aspects that are not supposed to represent what goes on in the mind.

All this does is remove the somewhat peculiar special status of simulation-based theory and return such theories to the arena occupied by all other scientific theories. Aspects of architectures are often unknown and are covered by explicit assumptions, which are subject to analysis. Important aspects of a total theory are often simply posited, such as the initial contents of the mind resulting from prior learning and external conditions, and behavior is invariably extremely sensitive to this. Analysis copes as best it can with such uncertainties, and the issue is not different with architectures.

3.6 Conclusions

An appropriate way to end this chapter is by raising some issues that reveal additional major steps required to pursue an adequate theory of the cognitive architecture. These questions have their roots in more

1 LINE SE
REG
1 LINE L

MASTER SET
Please Return

general issues of cognitive science, but our attention is on the implications for the cognitive architecture.

The list of requirements that could shape the architecture contains a number of items whose effects on the architecture we do not yet know, in particular the issues of acquiring capabilities through development, of living autonomously in a social community, and of exhibiting self-awareness and a sense of self (requirements 8 through 10).

Another issue is the effect on the architecture of its being a creation of biological evolution that grew out of prior structures shaped by the requirements of prior function. Thus we would expect the architecture to be shaped strongly by the structure of the perceptual and motor systems. Indeed we know from anatomical and physiological studies that vast amounts of the brain and spinal cord are devoted to these aspects. The question is what sort of architecture develops if it evolves out of the mammalian perceptual and motor systems, existing as sophisticated controllers but not yet fully capable of the flexibility that comes from full programmability. Beneath the level of the organization of the perceptual and motor systems, of course, is their realization in large, highly connected neural circuits. Here with the connectionist efforts (chapter 4) there is a vigorous attempt to understand what the implications are for the architecture.

An analogous issue is the relationship of emotion, feeling, and affect to cognition. Despite recent stirrings and a long history within psychology (Frijda 1986), no satisfactory integration yet exists of these phenomena into cognitive science. But the mammalian system is clearly constructed as an emotional system, and we need to understand in what way this shapes the architecture, if indeed it does so at all.¹³

We close by noting that the largest open issue with respect to the architecture in cognitive science is not all these phenomena whose impact on the architecture remains obscure. Rather it is our almost total lack of experience in working with complete cognitive architectures. Our quantitative and reasonably precise theories have been narrow; our general theories have been broad and vague. Even where we have approached a reasonably comprehensive architecture (Act* being the principal existing example), working with it has been sufficiently arcane and difficult that communities of scientists skilled in its art have not emerged. Thus we know little about what features of an architecture account for what phenomena, what aspects of an architecture connect what phenomena with others, and how sensitive various explanations are to variations in the architecture. About the only experience we have with the uses for architectures described in section 3.5 is analysis using gross parameters.

Such understandings do not emerge by a single study or by many studies by a single investigator. They come from many people exploring and tuning the architecture for many different purposes until derivations of various phenomena from the architecture become standard and

1 LINE SHORT
REGULAR
1 LINE LONG

understood. They come, as with so many aspects of life, from living them.

Notes

1. This technical use of the term is often extended to include combinations of software and hardware that produce a system that can be programmed. This broad usage is encouraged by the fact that software systems often present a structure that is meant to be fixed, so that the computer plus software operates just as if it were cast in hardware. In this chapter, however, we always take *architecture* in the narrow technical sense.
2. In part this is because functions are conceptual elements in an analysis of natural systems, and so what functions exist depends on the scheme of analysis.
3. They have been called *physical symbol systems* (Newell and Simon 1976) to emphasize that their notion of symbol derives from computer science and artificial intelligence, in contradistinction to the notion of symbol in the arts and humanities, which may or may not prove to be the same. The shorter phrase will do here.
4. Note, however, that it has proved functional in existing computers to drive the independence down as far as possible, to the bit.
5. Access structures can be (and are in plenitude) built up within the software of a system; we discuss the basic capability in the architecture that supports all such software mechanisms.
6. Note that the term *symbol* is used here for a type of structure and mechanism within a symbol system and not, as in *to symbolize*, as a synonym for something that represents. This notion of symbol, however, does at least require internal representation—addresses designate memory structures, input stimuli must map to fixed internal structures, and operator codes designate operations.
7. Conceivably this could be extended to include the external world as a distal level in the system's memory hierarchy (beyond the tertiary level). Symbol tokens would specify addresses in the external world, and the access and retrieval paths would involve perceptual and motor acts.
8. As the famous results of Turing, Church, and others have shown, this limit does not include all possible functional dependencies but only a large subclass of them, called the *computable functions*.
9. Another way to view the relationship of working memory to the long-term declarative memory is as two manifestations of a single underlying declarative memory. Each element in this underlying memory has two independently settable bits associated with it: whether the element is active (determines whether it is in working memory), and whether it is permanent (determines whether it is in long-term declarative memory).
10. In Anderson 1983 Act* is described as having two additional methods for creating new productions—generalization and discrimination—but they were later shown to be unnecessary (Anderson 1986).
11. *Virtual addressing* is a mechanism that introduces a small fixed amount of context, namely, a base address.

1 LINE SH
REGU
1 LINE L

12. As always, however, there is a trade-off. Recognition schemes are less flexible compared with location-pointer schemes, which are a genuinely task-independent medium for constructing accessing schemes, and hence can be completely adapted to the task at hand.

13. Feelings and emotions can be treated as analogous to sensations so they could affect the content of the cognitive system, even to including insistent signals, but still not affect the shape of the architecture.

References

Agrawal, D. P. 1986. *Advanced Computer Architecture*. Washington, DC: Computer Society Press.

Anderson, J. R. 1976. *Language, Memory and Thought*. Hillsdale, NJ: Lawrence Erlbaum.

Anderson, J. R. 1983. *The Architecture of Cognition*. Cambridge, MA: Harvard University Press.

Anderson, J. R. 1986. Knowledge compilation: The general learning mechanism. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds. *Machine Learning: An Artificial Intelligence Approach* Vol. 2. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Anderson, J. R., and Bower, G. 1973. *Human Associative Memory*. Washington, DC: Winston.

Anderson, J. R., and Thompson, R. 1988. Use of analogy in a production system architecture. In S. Vosniadou and A. Ortony, eds. *Similarity and Analogical Reasoning*. New York: Cambridge University Press.

Bell, C. G., and Newell, A. 1971. *Computer Structures: Readings and Examples*. New York: McGraw-Hill.

Brachman, R. J. 1979. On the epistemological status of semantic networks. In N. V. Findler, ed. *Associative Networks: Representation and Use of Knowledge by Computers*. New York: Academic Press.

Broadbent, D. E. 1954. A mechanical model for human attention and immediate memory. *Psychological Review* 64:205.

Brown, A. L. 1978. Knowing when, where, and how to remember: A problem in meta-cognition. In R. Glaser, ed. *Advances in Instructional Psychology*. Hillsdale, NJ: Erlbaum.

Card, S., Moran, T. P., and Newell, A. 1983. *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Erlbaum.

Clocksin, W. F., and Mellish, C. S. 1984. *Programming in Prolog* 2nd ed. New York: Springer-Verlag.

Craik, F. I. M., and Lockhart, R. S. 1972. Levels of processing: A framework for memory research. *Journal of Verbal Learning and Verbal Behavior* 11:671-684.

1 LINE SHORT
REGULAR
1 LINE LONG

REF SET
Return

Feigenbaum, E. A., and Simon, H. A. 1984. EPAM-like models of recognition and learning. *Cognitive Science* 8:305-336.

Fernandez, E. B., & Lang, T. 1986. *Software-Oriented Computer Architecture*. Washington, DC: Computer Society Press.

Frijda, N. H. 1986. *The Emotions*. Cambridge, Engl.: Cambridge University Press.

Gajski, D. D., Milutinovic, V. M., Siegel, H. J., and Furht, B. P. 1987. *Computer Architecture*. Washington, DC: Computer Society Press.

Hayes-Roth, B. 1985. A blackboard architecture for control. *Artificial Intelligence* 26:251-321.

Hopcroft, J. E., and Ullman, J. D. 1979. *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley.

Klahr, D. 1989. Information processing approaches to cognitive development. In R. Vasta, ed. *Annals of Child Development*. Greenwich, CT: JAI Press, pp. 131-183.

Laird, J. E., Newell, A., & Rosenbloom, P. S. 1987. Soar: An architecture for general intelligence. *Artificial Intelligence* 33(1):1-64.

Laird, J. E., Rosenbloom, P. S., and Newell, A. 1986. Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning* 1:11-46.

Maes, P., and Nardi, D., eds. 1988. *Meta-Level Architectures and Reflection*. Amsterdam: North-Holland.

Minsky, M. 1967. *Computation: Finite and infinite machines*. Englewood Cliffs, NJ: Prentice-Hall.

Minsky, M. 1975. A framework for the representation of knowledge. In P. Winston, ed. *The Psychology of Computer Vision*. New York: McGraw-Hill.

Newell, A. 1973. Production systems: Models of control structures. In W. C. Chase, ed. *Visual Information Processing*. New York: Academic Press.

Newell, A. 1980. Physical symbol systems. *Cognitive Science* 4:135-183.

Newell, A. 1987 (Spring). Unified theories of cognition. The William James lectures. Available in videocassette, Psychology Department, Harvard University, Cambridge, MA.

Newell, A., and Rosenbloom, P. S. 1981. Mechanisms of skill acquisition and the law of practice. In J. R. Anderson, ed. *Cognitive Skills and their Acquisition*. Hillsdale, NJ: Erlbaum.

Newell, A., and Simon, H. A. 1972. *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall.

Newell, A., and Simon, H. A. 1976. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM* 19(3):113-126.

1 LINE SHORT
REGULAR
1 LINE LONG

MASTER SET
Please Return

Polk, T. A., and Newell, A. 1988 (August). Modeling human syllogistic reasoning in Soar. In *Proceedings Cognitive Science Annual Conference—1988*. Cognitive Science Society. pp. 181-187.

Rosenbloom, P. S., and Newell, A. 1986. The chunking of goal hierarchies: A generalized model of practice. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds. *Machine Learning: An Artificial Intelligence Approach*. Vol. 2. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Rosenbloom, P. S., and Newell, A. 1988. An integrated computational model of stimulus-response compatibility and practice. In G. H. Bower, ed. *The Psychology of Learning and Motivation*. Vol. 21. New York: Academic Press.

Rosenbloom, P. S., Laird, J. E., and Newell, A. 1988. The chunking of skill and knowledge. In H. Bouma and B. A. G. Elsendoorn, ed. *Working Models of Human Perception*. London: Academic Press. pp. 391-440.

Schneider, W., and Shiffrin, R. M. 1977. Controlled and automatic human information processing: I. Detection, search and attention. *Psychological Review* 84:1-66.

Shiffrin, R. M., and Schneider, W. 1977. Controlled and automatic human information processing: II. Perceptual learning, automatic attending, and a general theory. *Psychological Review* 84:127-190.

Siewiorek, D., Bell, G., and Newell, A. 1981. *Computer Structures: Principles and Examples*. New York: McGraw-Hill.

Steele, G. L., Jr., ed., with contributors Fahlman, S. E., Gabriel, R. P., Moon, D. A., and Weinreb, D. L. 1984. *Common Lisp: The Language*. Marlboro, MA: Digital Press.

Steier, D. E., Laird, J. E., Newell, A., Rosenbloom, P. S., Flynn, R. A., Golding, A., Polk, T. A., Shivers, O. G., Unruh, A., and Yost, G. R. 1987 (June). Varieties of learning in Soar: 1987. In *Proceedings of the Fourth International Workshop on Machine Learning*. Los Altos, CA: Morgan Kaufman.

ENCLOSURE

VanLehn, K., ed. 1989. *Architectures for Intelligence*. Hillsdale, NJ: Erlbaum.

von Cranach, M., Foppa, K., Lepinies, W., and Ploog, D., eds. 1979. *Human Ethology: Claims and Limits of a New Discipline*. Cambridge, Engl.: Cambridge University Press.

1 LINE SHORT
REGULAR
1 LINE LONG